МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА БАГАТОПЛАТФОРМНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ КАТАЛОГОМ ДОКУМЕНТІВ»**

Виконав: студент 4 курсу, групи 6.1219-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
С. В. Магарян
(ініціали та прізвище)
Керівник декан математичного факультету, професор, д.т.н. Гоменюк С. І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С. М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ 07 ” 02 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Магаряну Станіславу Володимировичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка багатоплатформної інформаційної системи управління каталогом документів

Керівник роботи Гоменюк Сергій Іванович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

Затверджені наказом ЗНУ від « 26 » січня 2023 р. № 102-с

2. Строк подання студентом роботи 07.06.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.
3. Реалізація програмного застосунку.

4. Зміст роботи(перелік питань, які потрібно розробити) розглянути проблему створення багатоплатформних програм та реалізувати відповідну інформаційну систему управління каталогом документів

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 07.02.2023

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.02.2023	
2.	Вивчення загальної теорії розробки багатоплатформових застосунків	08.03.2023	
3.	Розробка структури бази даних.	01.04.2023	
4.	Написання програмного застосунку та його налагодження.	15.04.2023	
5.	Оформлення і нормоконтроль	01.05.23	
6.	Захист кваліфікаційної роботи.	2__.06.2023	

Студент _____ С. В. Магарян
(підпис) (ініціали та прізвище)

Керівник роботи _____ С. І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А. В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка багатоплатформної інформаційної системи управління каталогом документів»: 50 с., 11 рис., 42 джерела, 1 додаток.

БІБЛІОТЕКА QT, ВІРТУАЛЬНА МАШИНА, ІНТЕРПРЕТАТОР, ІНФОРМАЦІЙНА СИСТЕМА, КОМПІЛЯТОР, КРОСПЛАТФОРМНИЙ ЗАСТОСУНОК.

Об’єкт дослідження – багатоплатформні застосунки.

Предмет дослідження – способи розробки багатоплатформних застосунків.

Метод дослідження – практичний.

Метою кваліфікаційної роботи є розробка багатоплатформної інформаційної системи управління каталогом документів.

Кваліфікаційну роботу присвячено дослідженню способів розробки багатоплатформних застосунків для всіх основних сучасних операційних систем: Windows, Linux та macOS. Розглянуто основні принципи створення таких програм: з використанням інтерпретованих мов програмування; віртуальних машин (JVM) та бібліотеки Qt. Досліджено системи програмування, придатні для створення кросплатформних програм. Розглянутий теоретичний матеріал застосовано для практичної розробки багатоплатформного застосунку.

SUMMARY

Bachelor`s Qualifying Paper «Development of a Multi-platform Information System for Document Catalog Management»: 50 pages, 11 figures, 42 references, 1 appendice.

QT LIBRARY, VIRTUAL MACHINE, INTERPRETER, INFORMATION SYSTEM, COMPILER, CROSS-PLATFORM APPLICATION.

The object of research – is multiplatform applications.

The subject of research – is methods of developing multi-platform applications.

The method of research – is practical.

The aim of the qualification work is the development of a multi-platform information system for managing the document catalog.

The qualification work is devoted to researching methods of developing multi-platform applications for all modern operating systems – Windows, Linux and macOS. The main principles of creating such programs are considered: using interpreted programming languages; virtual machines (JVM) and the Qt library. Programming systems suitable for creating cross-platform programs have been studied. The considered theoretical material is applied for the practical development of a multi-platform application.

ЗМІСТ

ВСТУП

Багатоплатформністю (або кросплатформністю) називається властивість програмного забезпечення працювати без переробки (принаймні істотної) у багатьох програмних або апаратних платформах. Тобто, якщо деяка програма може одночасно працювати, наприклад, у середовищі операційних систем Windows та Linux, то вона вважається багатоплатформною (програмна багатоплатформність). Іншим прикладом кросплатформності є, наприклад, можливість запуску програми на різних типах несумісних процесорів (апаратна багатоплатформність).

Оскільки на сьогодні створено дуже велику кількість різноманітних апаратних платформ на базі таких процесорів, як AMD, Apple, Intel, Qualcomm, MediaTek, Samsung та ін., які працюють під управлінням різноманітних операційних систем (Android, iOS, Linux, macOS, Windows тощо), то проблема зниження собівартості розробки програмного забезпечення та його уніфікації для різних типів апаратних та програмних платформ на сьогодні є дуже актуальною.

Основними на сьогодні способами досягнення багатоплатформності програмного забезпечення є використання:

1) відповідних мов програмування високого рівня;
2) спеціалізованих середовищ виконання (віртуальних машин);
3) кросплатформних бібліотек, які дозволяють створювати незалежний від апаратної або програмної платформи код.

Найбільш очевидним та поширеним способом реалізації кросплатформності є застосування мов програмування високого рівня, які початково створювалися як машинно-незалежні. Ці мови можна поділити на дві великі групи: компільовані та інтерпретовані.

На перший погляд, використання компільованих мов програмування високого рівня (Fortran, C/C++, Go, Rust тощо) повинно автоматично

вирішувати проблему створення багатоплатформних програм, оскільки їх трансляція у конкретному апаратному або програмному середовищі повинна давати необхідний результат. Але, на жаль, це можливе лише при застосуванні обмеженого кола стандартних бібліотек. На практиці більшість програм не можуть бути написані без використання специфічних програмних (системних) або апаратних прикладних програмних інтерфейсів (API – Application Programming Interface), що не дозволяє, наприклад, за допомогою простої перекомпіляції переносити код між різними платформами.

Використання інтерпретованих мов програмування (PHP, JavaScript, Python, Ruby, Julia і т. п.) має певні переваги при створенні кросплатформного програмного забезпечення, оскільки вони, на відміну від компільованих мов, фактично виконуються не безпосередньо на цільові машині, а у середовищі спеціалізованої програми – інтерпретатора цієї мови. Наявність проміжного шару між платформою та програмою дозволяє на практиці у деяких випадках досягти багатоплатформності. Найбільш яскравим прикладом тут є мова Python. Проте використання інтерпретованих мов програмування не є панацеєю, оскільки швидкість їх виконання є набагато меншою, ніж у компільованих. Крім того, без наявності на цільовій платформі відповідного інтерпретатора запуск такої програми буде неможливим. До того ж інтерпретатор споживає багато системних ресурсів, що робить реалізацію такого підходу нераціональною (а інколи і неможливою) у випадку використання спеціалізованих комп’ютерів (бортових, SBC (single-board computer), контролерів тощо).

Іншим підходом до практичної реалізації багатоплатформності є використання спеціалізованих кросплатформних мов програмування (Java, C#) і віртуальних машин, в середовищі яких вони виконуються. Найбільш відомим прикладом тут є мова програмування Java і віртуальне середовище її виконання – Java Virtual Machine (JVM). Слід зазначити, що JVM виконує не програму, безпосередньо написану на мові Java, а деякий машинно-незалежний байт-код, отриманий шляхом компіляції початкового коду на мові

Java. Фактично JVM є інтерпретатором Java байт-коду, що має такі ж самі недоліки, як і використання інтерпретаторів інших мов програмування.

Ще одним поширеним підходом до розробки багатоплатформних програм, є використання спеціалізованих кросплатформних бібліотек, які уніфіковано реалізують необхідний функціонал для створення програм для роботи в різних платформах. Найбільш відомим прикладом такої бібліотеки є Qt, яка початково створювалася для реалізації високоякісного графічного інтерфейсу користувача для роботи у всіх сучасних операційних системах, але на сьогодні підтримує багато різного іншого функціоналу (вебпрограмування, роботу з мережею, графікою, базами даними і таке інше). Також можна виділити багатоплатформні бібліотеки для реалізації графічного інтерфейсу GTK+, FLTK, бібліотеку для роботи з тривимірною графікою OpenGL, бібліотеку широкого призначення Boost тощо.

Отже проблема створення кросплатформного програмного забезпечення має багато аспектів і потребує подальших досліджень.

Метою кваліфікаційної роботи є вивчення сучасних підходів до створення багатоплатформного програмного забезпечення та реалізація відповідної інформаційної системи управляння каталогом документів.

Дипломна робота складається з трьох розділів. У першому розділі розглянуто основні інструменти створення багатоплатформного програмного забезпечення: високорівневі мови програмування, віртуальні машини та багатоплатформні фреймворки і бібліотеки.

У другому розділі описано кросплатформну бібліотеку Qt. Особливу увагу приділено розгляду можливостей Qt для роботи з базами даних.

У третьому розділі описано розроблену із застосуванням Qt багатоплатформну інформаційну системи управління каталогом документів та наведено приклади її роботи у середовищі операційних систем Windows та Linux.

1 ОГЛЯД ОСНОВНИХ ІНСТРУМЕНТІВ СТВОРЕННЯ БАГАТОПЛАТФОРМНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Мови програмування високого рівня

Перший у світі програмований комп’ютер було побудовано у США у 1945 р. Він отримав назву ENIAC (Electronic Numerical Integrator and Computer) і використовувався для військових цілей [1]. Вже через вісім років у 1953 р. у Великий Британії було створено перший комерційний комп’ютер Leo [1]. Ера масового програмування фактично розпочалася на при кінці сорокових років – початку п’ятдесятих років XX сторіччя. Проте перші програмісти для створення програм могли користуватися лише безпосередньо машинним кодом конкретного комп’ютеру, запис програм на якому складався з одиниць та нулів.

Машинний код було прийнято вважати мовою програмування першого покоління. Головною проблемою використання цих мов окрім високої складності написання на них програм було те, що комп’ютери різних виробників застосовували відмінні системи машинних команд, що вимагало повну переробку програми для перенесення її на іншу машину.

Мовами програмування другого покоління вважаються асемблери, які появилися на початку 1950-х років. Це також низькорівневі засоби написання програм, проте початковий код в них записується у більш зрозумілій для людини формі із застосуванням спеціальних операторів. Для виконання такої програми на комп’ютері необхідна її трансляція у машинний код, для чого використовувалися спеціальні програми, які також отримали назву асемблерів [2]. Використання асемблерів також не вирішувало проблему кросплатформності, воно лише спрощувало процес програмування.

Для усунення цього недоліку у 1957 р. була розроблена перша в світі мова програмування високого рівня FORTRAN (FORmula TRANslator –

перекладач формул) [3]. Початковий код на цій мові записувався у більш зручній і зрозумілій для людини формі. Для виконання програми, написаної на мові високого рівня, необхідно застосувати спеціальну програму – транслятор, яка переводить початковий код у машинні команди цільової платформи.

На перший погляд, оскільки мови програмування високого рівня є абстрактними, тобто не залежать від особливостей конкретної обчислювальної машини, то вони дозволяють створювати уніфіковані програми для всіх типів комп'ютерів, для яких розроблено відповідні транслятори. Але на сьогодні, на жаль, це не так. Будь-яка сучасна операційна система має графічний інтерфейс та є підтримує одночасне виконання декількох програм (задач або процесів). Крім того, кожна сучасна програма може складатися з декількох потоків виконання (thread), що виконуються паралельно. Підтримка цих можливостей напряму залежить від вживаною апаратної платформи. Так, наприклад, програмування в середовищі операційної системи Windows передбачає застосування її прикладного програмного інтерфейсу Windows API, який є несумісним з API Unix – POSIX, або API Linux – LSB. Крім того, не зважаючи на схожість Unix та Linux, POSIX та LSB також не в повній мірі сумісні, що не дозволяє в деяких випадках просто переносити програми з Unix на Linux.

Таким чином, написання програм на мовах програмування високого рівня, які використовують специфічні API апаратної або програмної платформи, ускладнює або взагалі робить неможливим перенос таких програм на іншу платформу навіть при наявності там відповідного компілятора.

Окремо слід розглянути інтерпретовані мови програмування. Інтерпретатор, на відміну від компілятора, не переводить весь початковий код у машинні команди, а крок за кроком виконує зчитує команди початкового коду й виконує їх [4]. Зазвичай інтерпретовані програми працюють набагато повільніше, ніж компільовані. Але використання інтерпретатора має і певні переваги. Фактично він є певним середовищем виконання, де працюють програми. Тобто він є проміжним шаром між програмою і процесором, що дає

можливість реалізувати в деяких випадках кросплатформність. Але, як вже зазначалося, відносно низька швидкодія інтерпретаторів та значні системні ресурси, які вони споживають, не дозволяють повністю вирішити питання створення кросплатформних програм із їх використанням.

Отже, завдяки лише використанню мов програмування високого рівня проблему створення кросплатформних програм вирішити не вдасться.

1.2 Віртуальні машини

Одним з найбільш поширених способів реалізації багатоплатформності є використання спеціальних віртуальних машин, в середовищі яких можуть уніфіковано виконуватися певні класи програм. Зазвичай віртуальну машину можна представити як деякий уявний комп'ютер з певним стандартним набором апаратних та програмних ресурсів і відповідними API для взаємодії з ними.

Одним з найбільш відомих типів віртуальної машини, які використовуються для реалізації багатоплатформності, є JVM (Java Virtual Machine) – віртуальна машина мови програмування Java [5]. Java-програма спочатку компілюються у проміжний байт-код, який при виконанні інтерпретується і виконується JVM на конкретній платформі, за рахунок чого й досягається багатоплатформність (рис. 1.1).

Однією з головних цілей, які ставилися при розробці мови Java, було збереження незалежності від архітектури та переносність [6]. В цілому цю мету було досягнуто. На сьогодні мова Java завдяки використанню JVM є важливим інструментом кросплатформної розробки. Програма, написана на мові Java, буде працювати у будь-якій апаратній чи програмній платформі, для якої реалізовано JVM, без будь-яких змін у початковому коді та перекомпіляції. Проте, як і у випадку застосування інших інтерпретованих мов, швидкість роботи Java-програм значно нижча у порівнянні з

компільованими мовами. Крім того, для роботи у цільовій платформі необхідна наявність JVM, яка, у свою чергу, також споживає багато системних ресурсів.

Ще одним прикладом реалізації віртуального середовища виконання є Microsoft .NET Framework [7] – програмна технологія фірми Microsoft, що призначена для створення не тільки звичайних програм, а й веб-застосунків. На відміну від JVM, .NET Framework підтримує багато мов програмування, хоча головною для неї є мова C# [8], яка багато в чому схожа на Java.

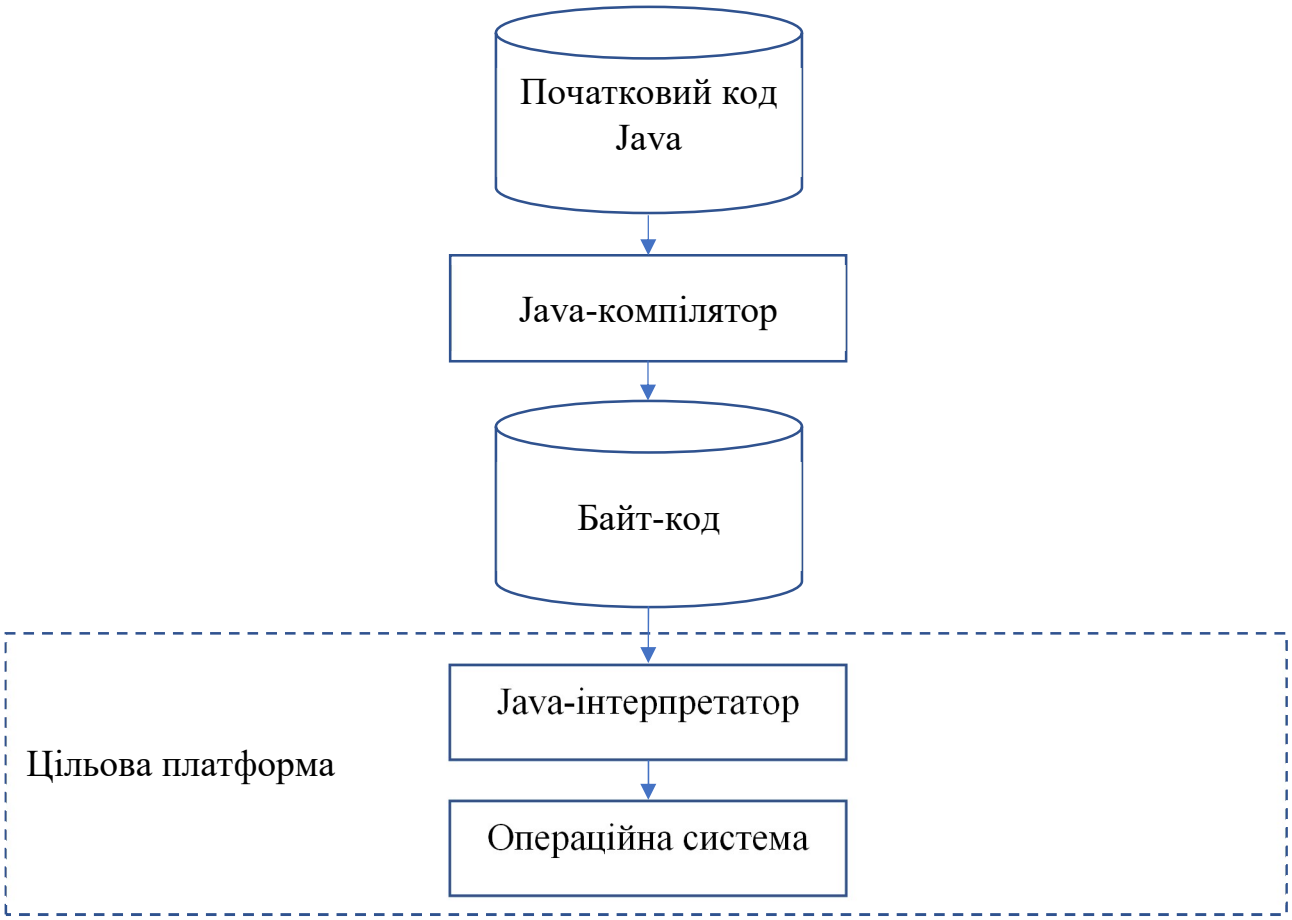

Рисунок 1.1 – Схема роботи віртуальної машини Java

Слід зазначити, що технологія Microsoft .NET Framework не досягла такого рівня поширеності, як JVM. Вона в основному використовується в середовищі операційних систем Windows, хоча і є спроби перенесення її на інші платформи. Самої відомою є MonoDevelop – кросплатформна IDE

(Integrated Development Equipment) для розробки програм на мовах C#, F# та ін. у середовищі операційних систем Linux, macOS та ін. [9].

1.3 Кросплатформні бібліотеки та фреймворки

Оскільки розробка багатоплатформних програмних застосунків дозволяє зменшити час розробки та кількість програмістів, задіяних до створення програмного забезпечення, а також охопити більшу номенклатуру пристроїв, на яких буде виконуватися програма, то цей підхід на сьогодні є доволі популярним і активно розвивається.

При огляді наявних багатоплатформних фреймоврків та бібліотек слід враховувати той факт, що їх можна умовно поділити на дві окремі категорії:

1) засоби, призначені для веб-розробки;
2) десктопні рішення.

Веб-розробка на сьогодні, де-факто, є прикладом кросплатформного програмування, оскільки веб-застосунки на стороні користувача виконуються в браузерах, що вимагає від програміста дотримання певних правил розробки. Хоча на серверній стороні веб-застосунки можуть істотно розрізнятися в залежності від платформи (наприклад, LAMP [10] та WAMP [11]).

Однією з найбільш старих та відомих багатоплатформних бібліотек для створення дво- та тривимірної графіки є OpenGL (Open Graphics Library) [12] – відкритий промисловий стандарт високоефективної комп'ютерної графіки, заснований провідними світовими IT-компаніями ще у 1992 р. OpenGL апаратно підтримується більшістю сучасних графічних відеоадаптерів, тому, якщо необхідно розробити багатоплатформну програму для роботи з графікою, використання OpenGL буде оптимальним. До того ж цей стандарт (бібліотека) підтримуються багатьма сучасними мовами програмування. Так, наприклад, мова JavaScript [13], яка активно використовується для веб-розробки, підтримує веб-версію OpenGL – WebGL [14].

Однією з найбільш складних задач, які доводиться вирішувати при розробці багатоплатформних програм, що працюють в середовищі операційних систем з графічним інтерфейсом, є створення уніфікованого інтерфейсу користувача (UI – User Interface), оскільки комп'ютерна графіка є найбільш апаратно-залежною складовою.

Тому достатньо велика кількість багатоплатформних бібліотек та фреймворків орієнтовані в першу чергу на створення саме графічного інтерфейсу користувача (GUI – Graphical User Interface). Однією з найбільш відомих таких бібліотек є GTK [15]. Це одна з найбільш популярних на сьогодні бібліотек для кросплатформної розробки елементів інтерфейсу для популярних сучасних операційних систем (Unix, Linux, Windows, macOS та ін.). Ця бібліотека була розроблена некомерційною спільнотою Gnome Foundation [16]. За її допомогою було написано багато реалізацій концепції робочого столу та віконних менеджерів (Gnome, Xfce, MATE, Cinnamon, AfterStep, Marco тощо), і такі популярні програми, як Chromium, FireFox, MonoDevelop, Gimp та інші. GTK написаний за допомогою мови програмування C [17], але існують форки для багатьох інших мов програмування, наприклад, C++ [18], Python [19], Java [5] та інші. Крім того GTK має спеціальний конструктор для візуальної розробки інтерфейсів – Glade [20].

Ще одним відомим прикладом багатоплатформного заосбу для створення графічного інтерфейсу користувача є бібліотека з відкритим початковим кодом FLTK (Fast Light Toolkit) [21]. С самого початку ця бібліотека створювалася для підтримки тривимірної графіки і тому має вбудований інтерфейс OpenGL, але вона добре підходить і для програмування звичайних графічних інтерфейсів.

FLTK базується на використанні власних незалежних від конкретної операційної системи віджетів, графіки та обробника подій, що дозволяє писати програми, які однаково виглядають і працюють на різних операційних системах. На відміну від багатьох інших подібних бібліотек, наприклад, GTK,

FLTK обмежується лише графічною функціональністю, що зумовлює її малий розмір і полегшує використання. Проте, головним недоліком FLTK вважається те, що програми, написанні за допомогою цієї бібліотеки, не завжди виглядають природно в середовищі конкретної операційної системи.

Для створення багатоплатформного графічного інтерфейсу програм, написаних на мові Python [19] використовується подіє-орієнтована графічна бібліотека Tkinter [22]. По суті вона є python-пакетом, призначеним для роботи з бібліотекою Tk, яка містить компоненти графічного інтерфейсу, реалізованого мовою програмування Tcl [23].

Кросплатформна бібліотека wxWidgets [24] – це набір інструментів з відкритим початковим кодом для розробки для створення, в першу, чергу, якісного графічного інтерфейсу користувача. Хоча, окрім роботи з GUI, вона містить набір класів для обробки графічних зображень; форматів HTML та XML; документів різних типів; архівів; файлових систем; процесів і багато іншого. wxWidgets підтримує роботу з мовами програмування C++ та Python, хоча є можливість викликати процедури wxWidgets і з інших мов. З використанням цієї бібліотеки написано такі програми, як FileZilla, AudaCity, BitTorrent та інші.

Ще одним прикладом багатоплатформного фреймворку є Avalonia [25]. Це досить новий засіб розробки, який використовує мову C# і платформу .NET Framework для кросплатформової розробки і створення інтерфейсів на базі XAML [26], завдяки якому Avalonia дозволяє створювати гнучкі та стилізовані інтерфейси.

Electron JS [27] – це JavaScript фреймворк, який дозволяє розробляти програмне забезпечення із застосуванням веб-технологій HTML, CSS та JS. Фактично Electron JS є двигуном Chromium [28], в якому і виконується весь програмний код. На сьогодні багато популярних програм написані на Electron JS: Slack, Discord, VSCode, Atom, Postman, Insomnia та ін. [27].

Крім вищезазначених слід відзначити ще низку багатоплатформних фреймворків, які активно використовуються в наш час.

JavaFX – фреймворк для розробки кросплатформних десктопних програм на мові Java, який підтримує реалізацію графічного інтерфейсу користувача, анімацію, мультимедіа та підтримку багатопотоковості [29].

Xamarin – фреймворк для розробки багатоплатформних десктопних додатків на C#. Він дозволяє створювати програми, які працюють в середовищі операційних систем Windows, macOS та Linux [30].

NativeScript [31] – фреймворк для розробки кросплатформових мобільних та десктопних програм на JavaScript [13], TypeScript [32] і Angular [33]. Він використовує нативні компоненти для створення інтерфейсу користувача і підтримує Windows, macOS і Linux.

OpenFL [34] – кросплатформна бібліотека для розробки ігор та програм, що працює в середовищі Windows, Linux, macOS, Android та iOS.

Flutter [35] – фреймворк для розробки кросплатформових мобільних та десктопних програм на Dart [36]. Він використовує свій власний віджетний двигун і підтримує Windows, MacOS, Linux, Android, iOS та Інтернет.

React Native [37] – це фреймворк для розробки кросплатформових мобільних та десктопних програм на JavaScript [13] і React [37]. Він використовує нативні компоненти для створення інтерфейсу користувача і підтримує Windows, macOS і Linux.

Ну і нарешті слід згадати фреймворк Qt [38] – потужний набір інструментів для створення кросплатформних додатків із застосуванням мов програмування C++ та Python (існують реалізації Qt для інших мов програмування, проте вони підтримуються виключно спільнотою). За допомогою Qt написані такі програми, як: Virtual Box, Skype, VLC Media Player, Opera, KDE та інші. Бібліотека Qt має власне середовище розробки – Qt Creator, яка включає програму Qt Designer для автоматизації створювати графічний інтерфейс. Значною перевагою Qt є наявність великої кількості документації, а також активну підтримку спільноти та безліч інших переваг. Саме застосуванню Qt й буде присвячено цю кваліфікаційну роботу.

2 ФРЕЙМВОРК QT

2.1 Огляд Qt

Qt [38] – це кросплатформний фреймворк, яка була написана за допомогою мови програмування C++ [18] і призначена для створення різноманітних програм із високоякісним графічним інтерфейсом користувача, що працюють у середовищі операційних систем сімейства Windows, Linux та maxOS.

Початково бібліотека Qt була розроблена у 1996 році і з того часу активно розвивається. Її використовують такі лідери IT-індустрії, як Autodesk, Google, Microsoft, Nokia, Panasonic, Siemens, Walt Disney Animation Studios та ін.

Крім створення графічного інтерфейсу користувача фреймворк Qt станом на сьогодні дозволяє:

– розробляти багатопотокові програми;

– створювати програми для роботи з комп’ютерною мережею;

– розробляти програми для роботи з дво- та тривимірною графікою (у тому числі і з використанням бібліотеки OpenGL [12]);

– працювати з базами даних, в т. ч. з використанням мови запитів SQL [39];

– працювати з XML [40];

– розробляти веб-застосунки;

– працювати з мультимедіа;

– будувати різноманітні діаграми та бізнес-графіку і багато іншого.

Бібліотека Qt містить велику кількість класів, які утворюють досить складну ієрархію (рис. 2.1).

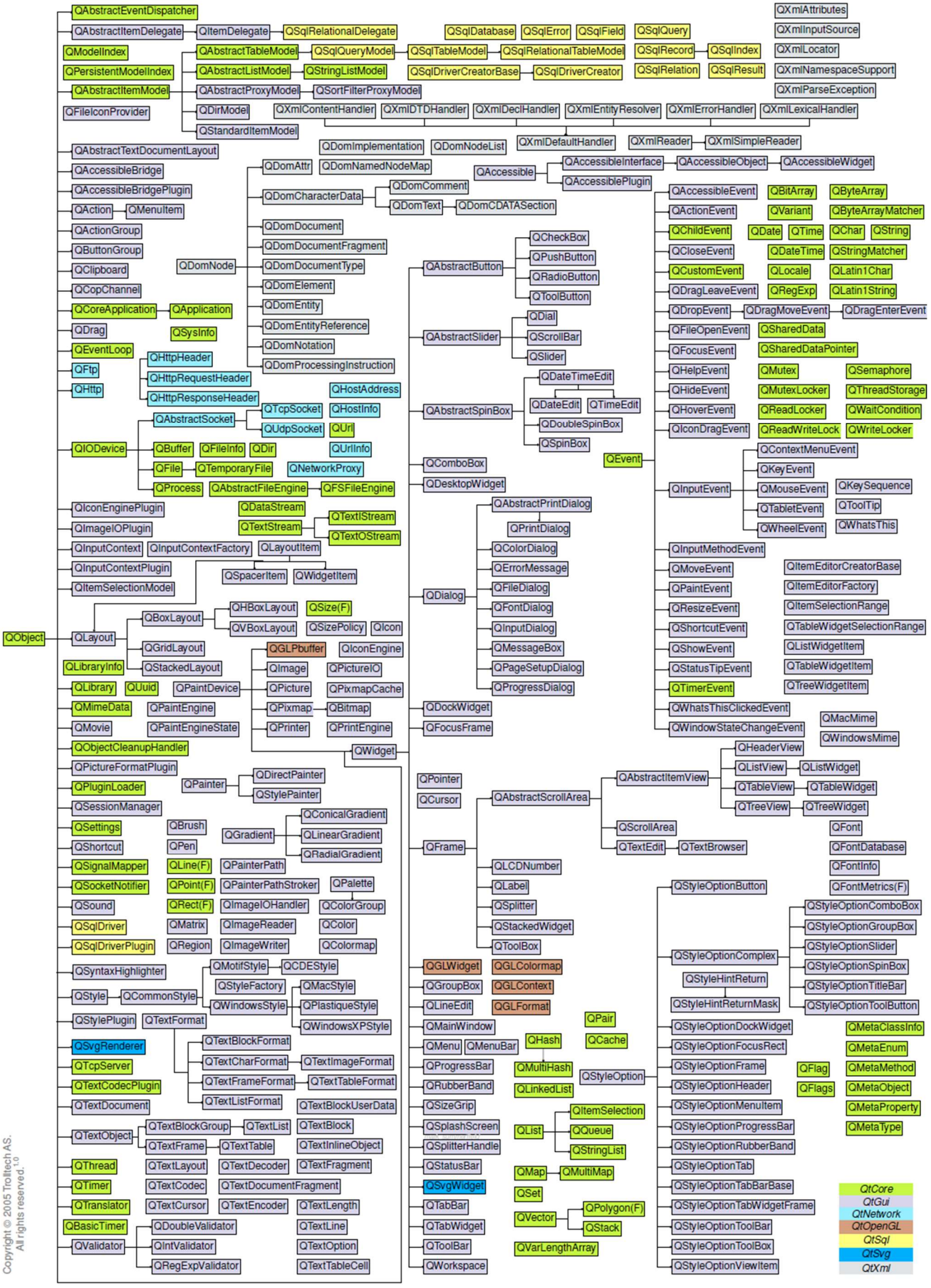

Рисунок 2.1 – Ієрархія класів Qt

За допомогою Qt написані такі відомі програми, як: KDE; Opera, Skype, Adobe Photoshop Album, Google Earth, VirtualBox, VCL media player і багато інших.

Широкою популярністю також користується кросплатформне середовище розробки Qt Creator, яке базується на використанні бібліотеки Qt і підтримує технологію візуального програмування (рис. 2.2).

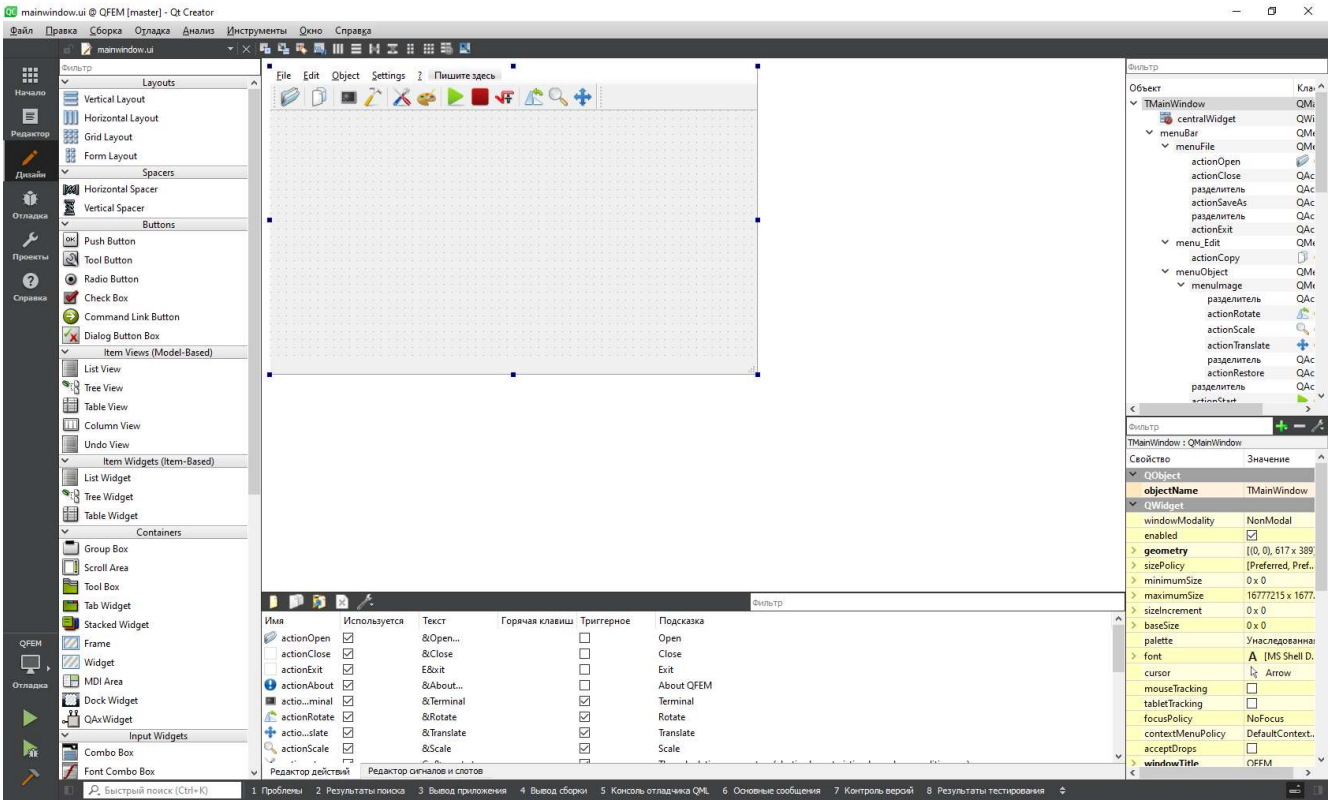

Рисунок 2.2 – Кросплатформне середовище розробки Qt Creator

Фреймворк Qt складається з понад 500 класів, які реалізують більшу частину функціоналу всіх сучасних операційних систем і для зручності використання об’єднані у модулі, основні з яких наведені у табл. 2.1.

Таблиця 2.1 – Основні модулі Qt

Модуль	**Назва**	**Призначення**
QtCore	core	Набір базових класів, не пов'язаних із GUI
QtGui	gui	Набір базових класів для програмування GUI

Продовження табл. 2.1

Модуль	Назва	Призначення
QtWidgets	widgets	Модуль, що містить графічні віджети для GUI
QtNetwork	network	Набір класів для програмування мережі
QtOpenGL	opengl	Модуль для програмування графіки OpenGL
QtSql	sql	Модуль для програмування баз даних
QtXml	xml	Набір класів для роботи з XML
QtMultimedia	mulimedia	Класи для роботи з мультимедіа
QtWebKit	webkit	Модуль для створення веб-застосунків
QtPrintSupport	printsupport	Модуль для роботи з принтером

Як видно з рис. 2.1 більшість класів бібліотеки Qt є нащадками від QObject. Базовий клас QObject підтримує наступні головні можливості:

– реалізацію механізму обробки повідомлень – сигнали та слоти (signal/slot);

– роботу таймером;

– механізми об’єднання об’єктів у ієрархії;

– події та механізми їх фільтрації;

– мета-об’єктну інформацію;

– приведення типів;

– властивості [41].

Для обміну інформацією про різноманітні події, що генеруються об’єктами в процесі їх функціонування, а також для їх обробки, в Qt використовується об’єктно-орієнтований механізм сигналів та слотів.

Використання таймера, визначеного в QObject, дозволяє класам-нащадкам застосовувати його у своїх цілях, економлячи при цьому час на розробку.

Механізми об’єднання об’єктів в ієрархії дозволяють скоротити часові витрати на розробку програм за рахунок автоматизації звільнення пам’яті для об’єктів, що створюються, оскільки базові об’єкти автоматично очистять пам’ять, займану своїми похідними об’єктами.

Механізм фільтрації подій дозволяє здійснювати їх перехоплення, завдяки чому можна змінити реакцію об’єктів на події, що відбуваються, без зміни вихідного коду класу.

Мета-об’єктна інформація містить дані про ієрархію класів, а також дозволяє дізнатися ім’я класу.

Приведення типів дозволяє з урахуванням мета-об’єктної інформації здійснювати перетворення типів між класами-нащадками від базового класу QObject.

Властивості – це спеціальні поля, для доступу до яких у класі мають існувати спеціальні методи читання. Властивості широко застосовуються у візуальному середовищі розробки GUI – Qt Designer [38].

2.2 Сигнали та слоти в Qt

Класичним підходом до програмної реалізації користувальницького інтерфейсу є застосування функцій зворотного виклику (callback function), які викликаються для обробки дій користувача з віджетами (елементами графічного інтерфейсу).

Цей підхід є достатньо старим і за своєю сутністю не є об’єктно-орієнтованим. Крім того його використання істотно ускладнює читання початкового коду.

У фреймворку Qt для обміну повідомленнями між об’єктами та їх обробки використовується власний цілком об’єктно-орієнтований механізм сигналів та слотів, за допомоги якого можна об’єднувати різноманітні об’єкти один з одним (рис. 2.3).

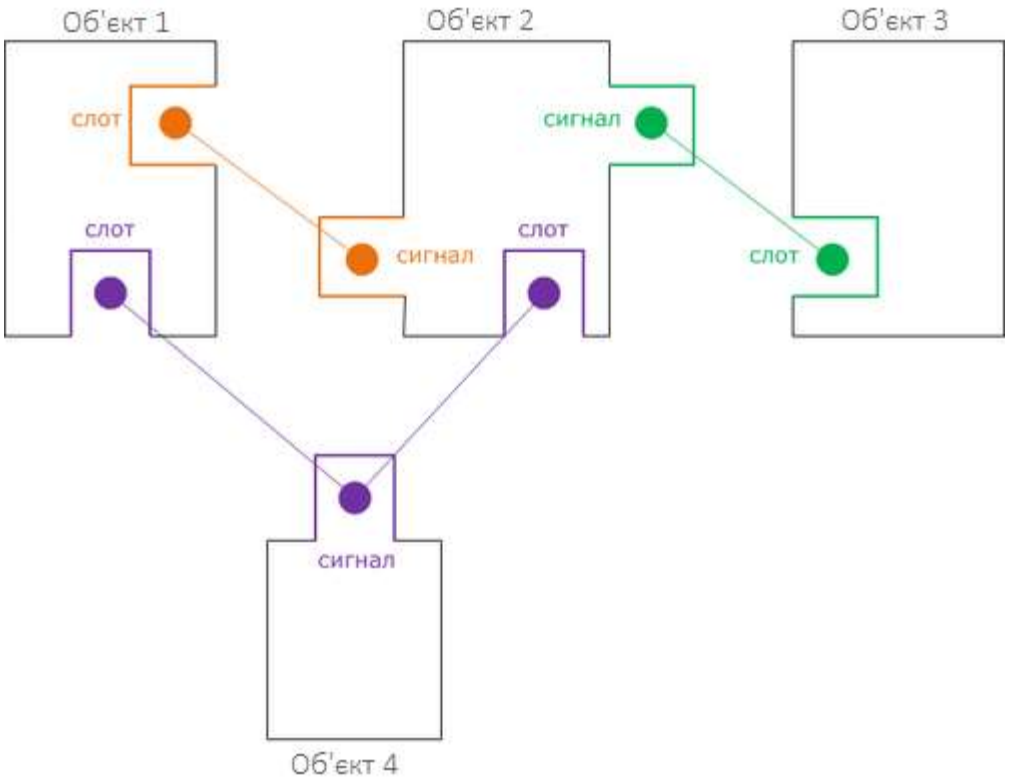

Рисунок 2.3 – Механізм сигналів та слотів

Застосування сигналів і слотів є основним концептом програмування з використанням фреймворку Qt. Кожен похідний від QObject клас може приймати і відправляти сигнали.

Так, наприклад, вбудований у об’єкт класу QObject таймер автоматично генерує та надсилає повідомлення про те, що час оновився. При цьому деякий віджет користувача, отримавши це повідомлення, може викликає відповідний метод (слот) для його обробки та оновлення свого вмісту (рис. 2.4).

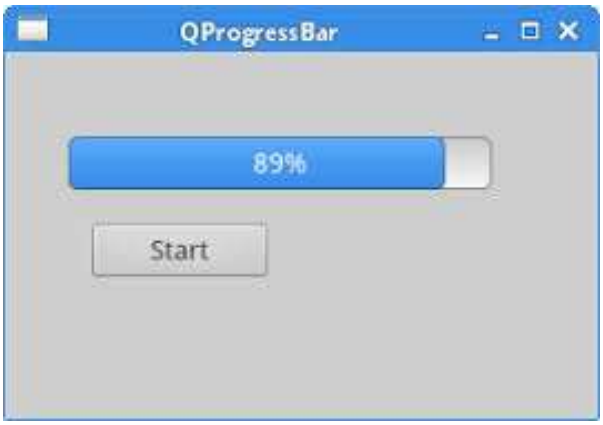

Рисунок 2.4 – Оновлення віджету при перехопленні сигналу від таймеру

В Qt сигнали – це спеціальні методи класів, які можуть пересилати повідомлення. Наприклад:

```
class TDrawScene: public QGLWidget
{
Q_OBJECT
// …
public:
void rotateX(int angle)
{
        emit xRotate(angle);        // Посилання сигналу
}
// …
signals:
        // Сигнали, що генерується при обертанні навколо осей координат
        void xRotation(int);
        void yRotation(int);
        void zRotation(int);
        // …
};
```

Слоти в Qt – це спеціальні методи класів, які автоматично викликаються при отриманні відповідного сигналу (для чого слот попередньо повинен бути з’єднаний із сигналом). Наприклад:

```
class TGLObject: public TGLMesh
{
        Q_OBJECT

// …
```

```
public slots:
        void mouseDoubleClickEvent(QMouseEvent*);

// …
};
```

З’єднання сигналів та відповідних слотів відбувається з використанням спеціального методу connect() класу QObject.

```
QMetaObject::Connection QObject::connect(const QObject *sender,
        const char *signal, const QObject *receiver, const char *method,
        Qt::ConnectionType type = Qt::AutoConnection);
```

де sender – вказівник на об’єкт, що надсилає сигнал;
signal – сигнал, з яким здійснюється з’єднання;
receiver – вказівник на об’єкт, що містить слот;
method –слот, який викликається;
type – режим обробки.

Наприклад:

```
QLabel *label = new QLabel();
QScrollBar *scrollBar = new QScrollBar();
QObject::connect(scrollBar, SIGNAL(valueChanged(int)),
                label, SLOT(setNum(int)));
```

Від’єднання сигналу від слоту відбувається автоматично при видаленні відповідного об’єкта або за допомогою спеціального методу disconnect() класу QObject.

```
bool QObject::disconnect( const char *signal = nullptr,
```

```
                                const QObject *receiver = nullptr,
                                const char *method = nullptr) const
```

Наприклад:

```
// Вимкнення всіх сигналів
disconnect(myObject, nullptr, nullptr, nullptr);
```

2.3 Приклади створення програм в Qt

Найпростішу програму з графічним інтерфейсом за допомогою фреймворку Qt можна, наприклад, написати таким чином.

```
#include <QtWidgets>

int main(int argc, char **argv)
{
   QApplication app(argc, argv);
   QMainWindow mw;

   mw.setWindowTitle("Hello world!");
   mw.show();
   return app.exec();
}
```

Тут клас QApplication інкапсулює поняття програмного застосунку, а клас QMainWindow – головне вікно програми.

Компіляція та запуск цієї програми приведуть до наступного результату (рис. 2.5).

Рисунок 2.5 – Найпростіша Qt-програма з графічним інтерфейсом

Для компіляції програм, що використовують Qt, необхідно створити спеціальний файл налаштувань (проєкту). У найпростішому випадку він може, наприклад, виглядати таким чином.

```
QT += gui widgets
CONFIG += c++17
SOURCES += main.cpp
```

Файли проєктів у Qt оброблюються за допомогою спеціальної утиліти qmake. Найчастіше у файлах проєктів використовуються такі параметри (табл. 2.2).

Таблиця 2.2 – Параметри файлу проєкту у Qt

Параметр	**Призначення**
HEADERS	Список заголовних файлів проекту
SOURCES	Список вихідних файлів проекту
TARGET	Ім'я файлу-результату
TEMPLATE	Тип результату (app – програма)
INCLUDEPATH	Шлях до заголовних файлів
LIBS	Шлях до бібліотек
CONFIG	Параметри компілятора
FORMS	Файли, що описують екранні форми
TRANSLATIONS	Задає файли перекладів для проекту

2.4 Бібліотека Tulip

Найбільш часто у програмуванні доводиться виконувати обробка одноманітних даних (колекцій), які організовані тим чи іншим, заздалегідь визначеним способом.

Для виконання таких маніпуляцій фреймворк Qt пропонує спеціальну бібліотеку контейнерів – Tulip, яка є частиною ядра Qt.

Класи, що утворюють Tulip, розташовані у модулі QtCore (табл. 2.1). Вони підтримують:

– контейнери;
– ітератори;
– алгоритми (рис. 2.6).

Рисунок 2.6 – Структура Tulip

Контейнерами прийнято називати спеціальні класи, що використовуються для зберігання елементів різних типів. Зазвичай контейнери реалізують функціонал спеціальних структур даних: списків, черг, стеків тощо.

У бібліотеці Qt всі контейнерні класи реалізовані у вигляді шаблонів, тобто вони можуть зберігати дані будь-яких типів. Усі контейнери поділяються на дві групи: послідовні та асоціативні.

Послідовні контейнери містять упорядковані колекції, де кожен елемент займає певну позицію. До них належать:

– QVector<T> (вектор);
– QList<T> (список);
– QLinkedList<T> (двозв’язний список);
– QStack<T> (стек);
– QQueue<T> (черга).

Асоціативні контейнери (колекції, де позиція елемента залежить від його значення):

– QSet<T> (множина);
– QMap<T> (словник);
– QMultiMap<T> (мультисловник);
– QHash<T> (хеш);
– QMultiHash<T> (мультихеш).

2.5 Засоби роботи з базами даний Qt

Робота з базами даних у Qt відбувається на трьох різних рівнях:

1. Рівень драйверів (низький) – містить класи QSqlDriver, QSqlDriverCreator, QSqlDriverCreatorBase, QSqlDriverPlugin та QSqlResult, які реалізують низькорівневий доступ до баз даних. Тобто на цьому рівні відбувається фізичне відкриття баз даних конкретних типів.

2. Рівень SQL [42] (середній) – базується на використанні класів QSqlDatabase, QSqlQuery, QSqlDatabase, QSqlError, QSqlField, QSqlIndex і QsqlRecord. Взаємодія з базами даних здійснюється із застосуванням мови SQL.

3. Рівень інтерфейсу користувача (високий) – складається з класів QSqlQueryModel, QSqlTableModel та QSqlRelationalTableModel і призначений

для зв’язування інформації з баз даних і дата-орієнтованими графічними віджетами.

Для підключення до бази даних за допомогою класів QSqlQuery та QSqlQueryModel необхідно створити та відкрити одне або декілька фізичних з’єднань. Qt може працювати з великою кількістю різноманітних баз даних (для з’єднання з конкретним типом необхідна наявність певного плагіна). Найбільш поширеними типами баз даних, для яких постачаються плагіни, є:

QMYSQL (система управління базами даних MySQL);

QODBC (Microsoft SQL Server та інші ODBC-сумісні бази даних);

QPSQL (PostgreSQL);

QSQLITE (SQLite) та інші.

Так, наприклад, для підключення до бази даних MySQL можна написати наступний код.

```
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL", "mydb");

db.setHostName("host");
db.setDatabaseName("userdb");
db.setUserName("user_name");
db.setPassword("qwerty");
bool ok = db.open();
```

Тут у першому рядку створюється об’єкт з’єднання, а останньому – він відкривається. В інших рядках коду здійснюється налаштування назви з’єднання, бази даних, вузла, а також логіна та пароля користувача.

Як тільки з’єднання встановлено, можна викликати статичну функцію QSqlDatabase::database() з будь-якого місця програми із зазначенням імені з’єднання, щоб отримати покажчик на це з’єднання.

Якщо виклик open() зазнає невдачі, він поверне false. У цьому випадку можна отримати інформацію про помилку, викликавши QSqlDatabase::lastError().

Для видалення з’єднання з базою даних, треба спочатку закрити базу даних за допомогою QSqlDatabase::close(), а потім видалити з’єднання за допомогою статичного методу QSqlDatabase::removeDatabase().

Для отримання інформації з баз даних та маніпулювання нею використовується клас QSqlQuery, який забезпечує інтерфейс для виконання SQL запитів та навігації з результуючої вибірки. Для виконання SQL запитів необхідно створити об’єкт QSqlQuery і викликати його метод QSqlQuery::exec(). Наприклад, це можна зробити таким чином.

```
QSqlQuery query;

query.exec("SELECT name, population FROM country WHERE population > 5");
```

Конструктор класу QSqlQuery приймає додатковий аргумент QSqlDatabase, який уточнює, яке з’єднання з базою даних використовується. Якщо його не вказати, то використовується стандартне з’єднання. У разі виникнення помилки, метод exec() повертає false. Отримати інформацію про проблему, що виникла, можна за допомогою методу QSqlQuery::lastError().

QSqlQuery надає одноразовий доступ до результуючої вибірки одного запиту. Після виклику exec(), внутрішній покажчик QSqlQuery вказує на позицію перед першим записом. Якщо викликати метод QSqlQuery::next() один раз, він перемістить покажчик до першого запису. Після цього необхідно повторювати виклик next(), щоб отримувати доступ до інших записів, доки він не поверне false. Наприклад.

```
while (query.next())
```

```
{
    QString name = query.value(0).toString();
    int population = query.value(1).toInt();
    qDebug() << name << population;
  }
```

Клас QSqlQuery може виконувати не тільки оператор SELECT мови SQL, а й будь-які інші команди.

Для відображення інформації з баз даних користувачу використовуються класи QListView, QTableView і QTreeView, які можуть використовувати джерела інформацію, які інкапсулюються класами QSqlQueryModel, QSqlTableModel і QSqlRelationalTableModel. На практиці найчастіше використовується QTableView у зв’язку з тим, що результуюча SQL вибірка, по суті, є двовимірною структурою даних. Наприклад.

```
QSqlTableModel model;

model.setTable("country");
QTableView *view = new QTableView;
view->setModel(&model);
view->show();
```

Таким чином, бібліотека Qt містить весь необхідний для роботи з базами даних, набір класів, що реалізує всі необхідні механізми по фізичному доступу до даних, маніпуляції з ними засобами мови SQL і наочного представлення їх користувачу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ БАГАТОПЛАТФОРМНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Структура бази даних

Будь-яка інформаційна система базується на певному сховищі даних, в якості якого частіше за все використовуються бази даних. Для зберігання інформації про каталог документів, яким оперую певна організація, пропонуються наступна структура бази даних (рис. 3.1).

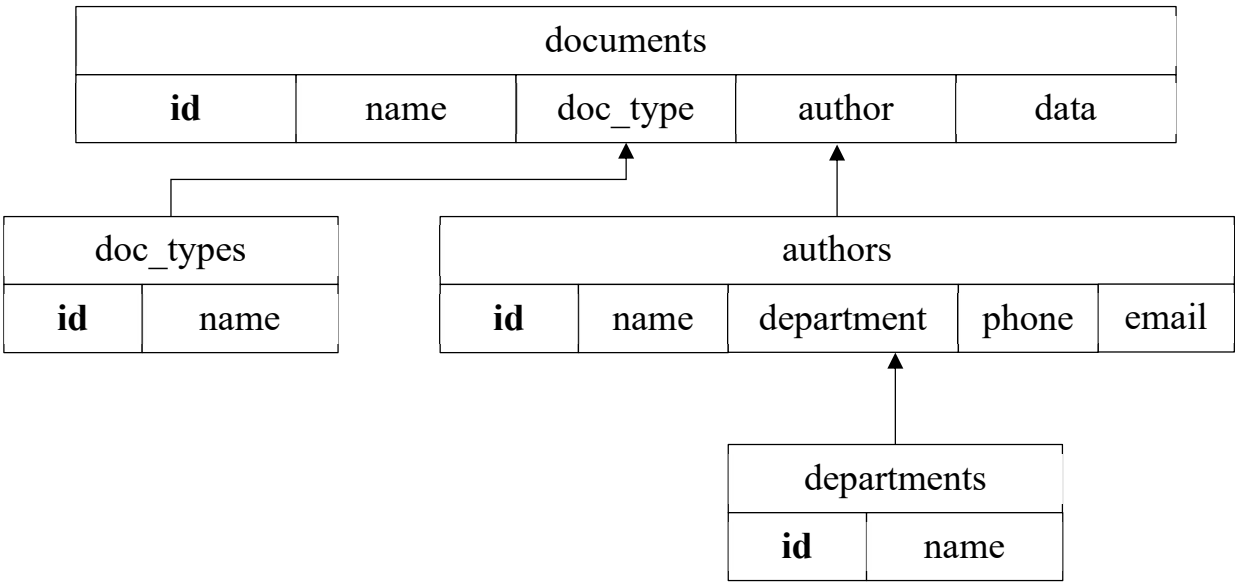

Рисунок 3.1 – Структура бази даних інформаційної системи управління документами

Тут головною таблицею є documents. Вона містить наступні поля (атрибути у термінах реляційної теорії баз даних):

– id – первинний ключ (INT);

– name – назва документу (TEXT);

– doc_type – зовнішній ключ, що посилається на таблицю doc_types, яка містить довідник типів документу (html, doc, pdf тощо) (INT);

– author – зовнішній ключ, що посилається на таблицю authors, яка містить список авторів документу (INT);

– data – вміст документу (BLOB).

Призначення та структура інших таблиць є достатньо очевидної за наведеної на рис. 3.1 структури.

3.2 Програмна реалізація додатку dbarch

Для досягнення поставленої мети – розробки багатоплатформної інформаційної системи управління каталогом документів із застосуванням фреймворку Qt і мови програмування C++ було реалізовано додаток dbarch. Початковий код класу MainWindow, що описує головне вікно, наведено в Додатку А. Для простоти реалізації використовувалася база даних SQLite.

При створенні об’єкту класу MainWindow викликається його метод openDB(), який перевіряє наявність бази даних. В разі її відсутності запускається процедура createDB(), яка створює нову базу даних і всі необхідні таблиці в ній.

Компіляція та запуск програми у середовищі операційної системи Windows приводять до наступного результату (рис. 3.2, 3.3).

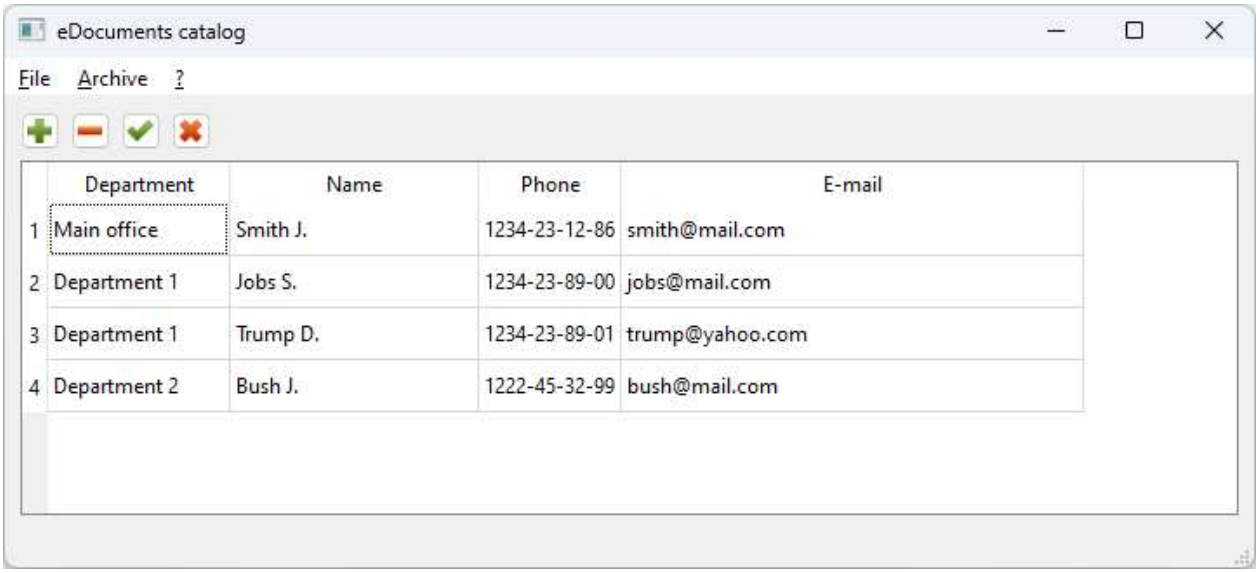

Рисунок 3.2 – Інтерфейс програми dbarch

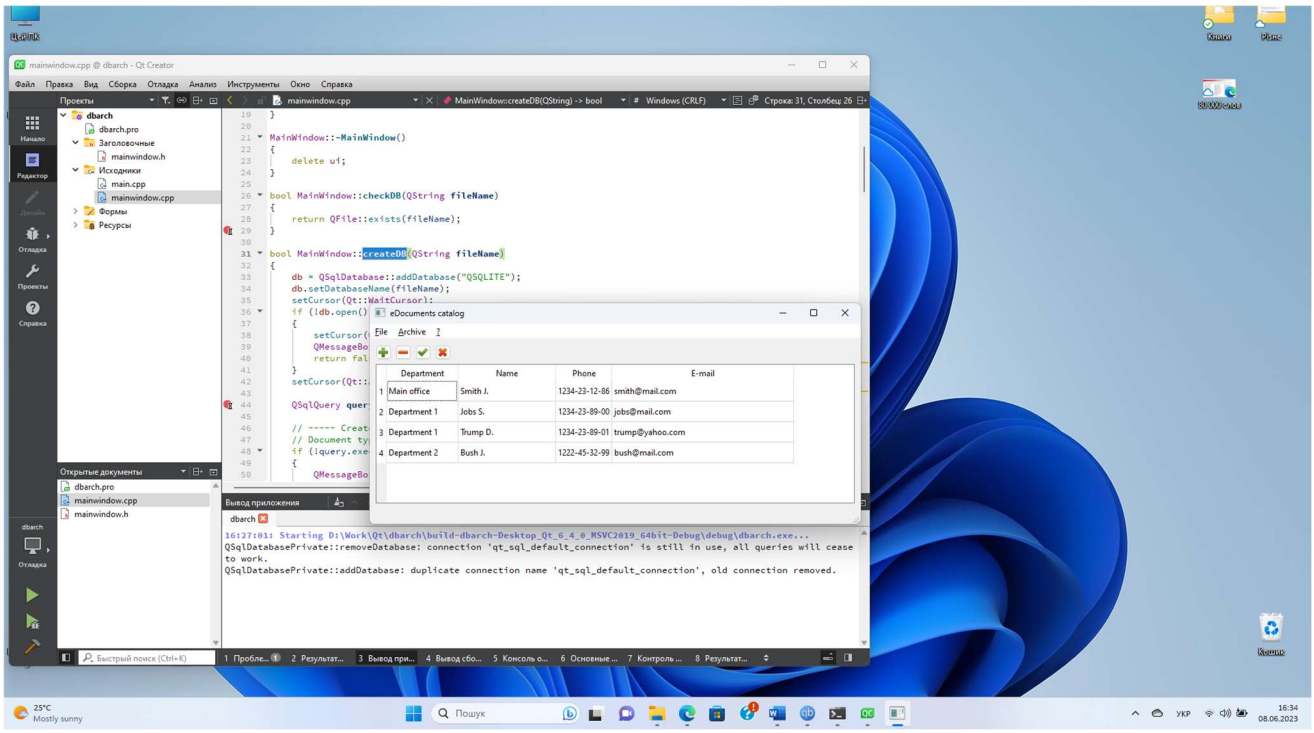

Рисунок 3.3 – Запуск програми dbarch в середовищі Windows

Результат компіляції та запуску додатку dbarch в Linux наведені на рис. 3.4.

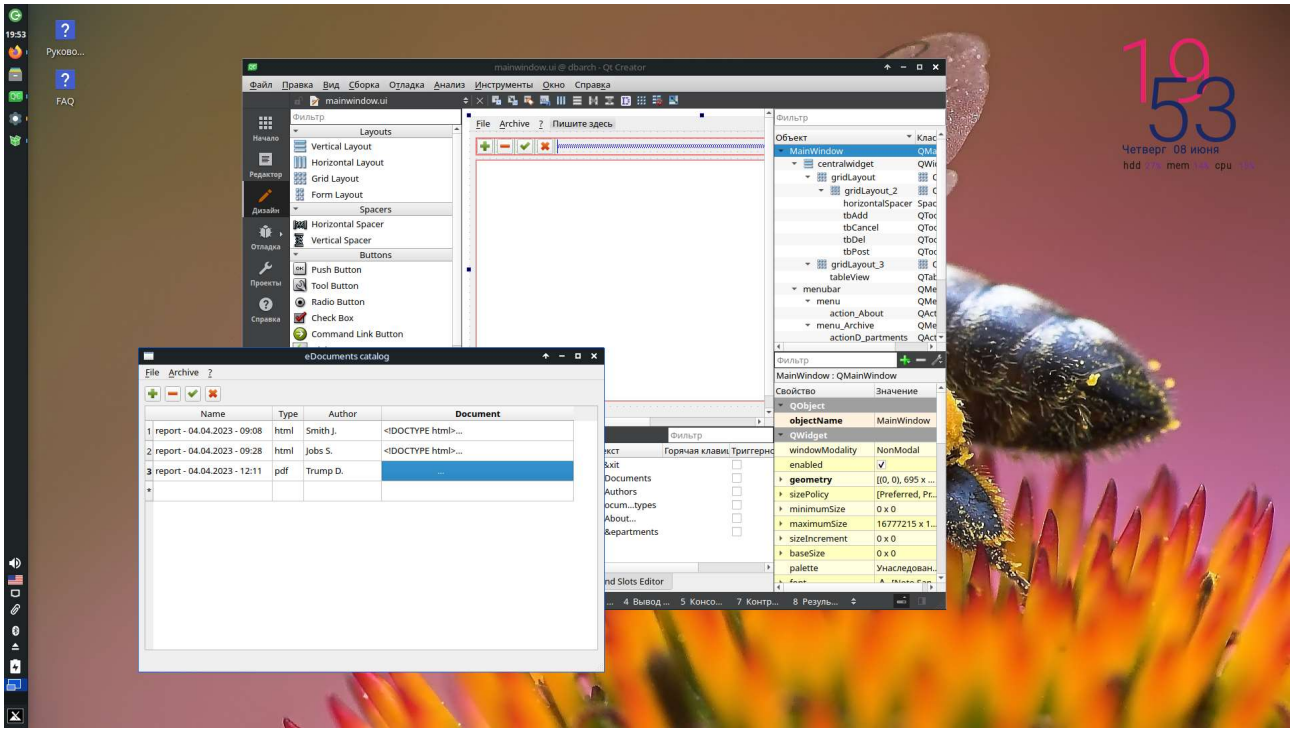

Рисунок 3.4 – Виконання програми dbarch в середовищі MX Linux

ВИСНОВКИ

Безумовно створення системної програми, наприклад, драйверу пристрою або утиліти низькорівневого доступу до апаратний можливостей комп’ютера багатоплатформним зробити практично неможливо. Проте розробку прикладних програм загального призначення можна і потрібно робити з використанням кросплатформного підходу. Це дає можливість, по-перше, зменшити кількість задіяних до розробки програмного забезпечення програмістів, а, по-друге, охопити більшу кількість пристроїв, на яких буде виконуватися програма.

Очевидно, що подібний принцип розробки програмного забезпечення не позбавлений і недоліків. Так кросплатформна програма при використанні віртуальних середовищ виконання, зазвичай працює більш повільно, ніж нативна, оскільки реалізувати для неї повноцінну апаратно-залежну оптимізацію важче, проте зі зростанням швидкодії сучасної обчислювальної техніки цей недолік стає все менш критичним.

Таким чином, розробка багатоплатформних програм на сьогодні є актуальною і важливою задачею і потребує подальших досліджень.

ПЕРЕЛІК ПОСИЛАНЬ

1. Rojas R., Hashagen U. The First Computers: History and Architectures. MIT Press, 2002. 471 p.

2. Дудзяний І. М., Черняхівський В. В. Програмування мовою асемблера. Львів : ЛНУ імені Івана Франка, 2002. 112 с.

3. Subrata R. Fortran 2018 With Parallel Programming. CRC Press, 2019. 654 p.

4. Hunter R. The essence of compilers. New York : Prentice Hall, 1998. 254 p.

5. Agarwal S., Bansal H. Java in Depth. BPB Publications, 2023. 345 p.

6. The Java Language Environment. URL: https://www.oracle.com/java/technologies/introduction-to-java.html (дата звернення 04.06.2023).

7. Abrams B. NET Framework Standard Library Annotated Reference. Addison-Wesley Professional, 2006. 560 p.

8. Olsson M. C# 10 Quick Syntax Reference: A Pocket Guide to the Language, APIs, and Library. Apress, 2022. 214 p.

9. MonoDevelop – Cross platform IDE for C#, F# and more. URL: https://www.monodevelop.com/ (дата звернення 04.06.2023).

10. LAMP (Linux, Apache, MySQL, PHP). URL: https://www.techtarget.com/whatis/definition/LAMP-Linux-Apache-MySQL-PHP (дата звернення 07.06.2023).

11. WAMPSERVER – plate-forme de développement Web sous Windows. URL: https://www.wampserver.com/ (дата звернення 04.06.2023).

12. OpenGL. The Industry's Foundation for High Performance Graphics. URL: https://www.opengl.org/ (дата звернення 04.06.2023).

13. Springer S. Node.js: The Comprehensive Guide to Server-Side JavaScript Programming. Rheinwerk Computing, 2022. 834 p.

14. WebGL: 2D and 3D graphics for the web. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API (дата звернення 05.06.2023).

15. The GTK Project – a free and open-source cross-platform widget toolkit. URL: https://www.gtk.org/ (дата звернення 07.06.2023).

16. The GNOME Foundation. URL: https://foundation.gnome.org/ (дата звернення 07.06.2023).

17. C Tutorial. URL: https://www.w3schools.com/c/index.php (дата звернення 07.06.2023).

18. C++ Introduction. URL: https://www.w3schools.com/cpp/cpp_intro.asp (дата звернення 07.06.2023).

19. Welcome to python.org. URL: https://www.python.org/ (дата звернення 07.06.2023).

20. Glade. URL: https://wiki.gnome.org/Apps/Glade (дата звернення 07.06.2023).

21. Fast Light Toolkit. URL: https://www.fltk.org/ (дата звернення 07.06.2023).

22. Moore A. D. Python GUI Programming with Tkinter: Design and build functional and user-friendly GUI applications. Packt Publishing, 2021. 664 p.

23. Flynt C. Tcl/Tk: A Developer's Guide. Morgan Kaufmann, 2012. 817 p.

24. wxWidgets: Cross-platform GUI Library. URL: https://www.wxwidgets.org/ (дата звернення 07.06.2023).

25. Avalonia UI – Home. URL: https://avaloniaui.net/ (дата звернення 07.06.2023).

26. XAML overview. URL: https://learn.microsoft.com/en-us/windows/uwp/xaml-platform/xaml-overview (дата звернення 07.06.2023).

27. Build cross-platform desktop apps with JavaScript, HTML, and CSS. URL: https://www.electronjs.org/ (дата звернення 07.06.2023).

28. The Chromium Projects. URL: https://www.chromium.org/chromium-projects/ (дата звернення 07.06.2023).

29. JavaFX. URL: https://openjfx.io/ (дата звернення 07.06.2023).

30. Xamarin. URL: https://dotnet.microsoft.com/en-us/apps/xamarin (дата звернення 07.06.2023).

31. NativeScript. URL: https://nativescript.org/ (дата звернення 07.06.2023).

32. TypeScript is JavaScript with syntax for types. URL: https://www.typescriptlang.org/ (дата звернення 07.06.2023).

33. Angular. URL: https://angular.io/ (дата звернення 07.06.2023).

34. OpenFL – Creative expression for the desktop, mobile, web and console platforms. URL: https://www.openfl.org/ (дата звернення 07.06.2023).

35. Flutter – Build apps for any screen. URL: https://flutter.dev/ (дата звернення 07.06.2023).

36. Dart programming language. URL: https://dart.dev/ (дата звернення 07.06.2023).

37. React Native · Learn once, write anywhere. URL: https://reactnative.dev (дата звернення 07.06.2023).

38. Qt | Tools for Each Stage of Software Development Lifecycle. URL: https://www.qt.io/ (дата звернення 07.06.2023).

39. SQL Tutorial. URL: https://www.w3schools.com/sql/ (дата звернення 07.06.2023).

40. XML Tutorial. URL: https://www.w3schools.com/xml/ (дата звернення 07.06.2023).

41. QObject Class. URL: https://doc.qt.io/qt-6/qobject.html (дата звернення 07.06.2023).

42. SQL Tutorial. URL: https://www.w3schools.com/sql/ (дата звернення 07.06.2023).

ДОДАТОК А

Початковий код класу MainWindow

```
// mainwidow.h
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QtSql/QSqlDatabase>
#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class QSqlRelationalTableModel;
//class QSqlDatabase;

class MainWindow : public QMainWindow
{
   Q_OBJECT

public:
   MainWindow(QWidget *parent = nullptr);
   ~MainWindow();

public slots:
   void slotAddRow(void);
   void slotDelRow(void);
```

```
    void slotSaveChanges(void);
    void slotCancelChanges(void);
    void slotOnDataChanged(const QModelIndex&, const QModelIndex&);
    void slotOpenDocuments(void);
    void slotOpenDepartments(void);
    void slotOpenDocTypes(void);
    void slotOpenAuthors(void);

private:
    Ui::MainWindow *ui;
    QSqlDatabase db;
    QSqlRelationalTableModel *model;
    bool checkDB(QString);
    bool createDB(QString);
    void openDB(void);
    void openTable(QString);
    void closeDB(void);
};
#endif // MAINWINDOW_H
```

```
// mainwidow.cpp
#include <QMessageBox>
#include <QtSql/QSql>
#include <QtSql/QSqlQuery>
#include <QtSql/QSqlError>
#include <QtSql/QSqlRelationalDelegate>
#include <QSqlRelationalTableModel>
#include <QFile>
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
   : QMainWindow(parent)
   , ui(new Ui::MainWindow)
{
   ui->setupUi(this);
   openDB();
   openTable("documents");

}

MainWindow::~MainWindow()
{
   delete ui;
}

bool MainWindow::checkDB(QString fileName)
{
   return QFile::exists(fileName);
}
```

```
bool MainWindow::createDB(QString fileName)
{
   db = QSqlDatabase::addDatabase("QSQLITE");
   db.setDatabaseName(fileName);
   setCursor(Qt::WaitCursor);
   if (!db.open())
   {
      setCursor(Qt::ArrowCursor);
      QMessageBox::critical(this, tr("Error"),tr("Error creating database!"),
QMessageBox::Ok);
      return false;
   }
   setCursor(Qt::ArrowCursor);

   QSqlQuery query;

   // ----- Create table -------
   // Document type
   if (!query.exec(QString("CREATE TABLE doc_types (id INTEGER NOT
NULL PRIMARY KEY AUTOINCREMENT, name       TEXT)")))
   {
      QMessageBox::critical(this, tr("Error"),tr("Error creating database!"),
QMessageBox::Ok);
      db.close();
      return false;
   }
   // Department
   if (!query.exec(QString("CREATE TABLE departments (id INTEGER
NOT NULL PRIMARY KEY AUTOINCREMENT, name TEXT)")))
```

```
    {
        QMessageBox::critical(this, tr("Error"),tr("Error creating database!"), QMessageBox::Ok);
        db.close();
        return false;
    }
//     if (!query.exec(QString("INSERT INTO departments (name) VALUES ('%1')").arg(tr("Main office"))))
//     {
//         QMessageBox::critical(this, tr("Error"),tr("Error creating database!"), QMessageBox::Ok);
//         db.close();
//         return false;
//     }
    // Authors
    if (!query.exec(QString("CREATE TABLE authors (id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, department INT, name TEXT, phone TEXT, email TEXT)")))
    {
        QMessageBox::critical(this, tr("Error"),tr("Error creating database!"), QMessageBox::Ok);
        db.close();
        return false;
    }
    // Documents
    if (!query.exec(QString("CREATE TABLE documents (id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, name TEXT, doc_type INT, author INT, data BLOB)")))
    {
```

```
        QMessageBox::critical(this, tr("Error"),tr("Error creating database!"),
QMessageBox::Ok);
        db.close();
        return false;
    }
    db.close();
    return true;
}

void MainWindow::openDB(void)
{
    db = QSqlDatabase::addDatabase("QSQLITE");
    // Проверка наличия БД
    if (!checkDB("dbarch.db"))
        if (!createDB("dbarch.db"))
            return;
    db.setDatabaseName("dbarch.db");

    setCursor(Qt::WaitCursor);
    if (!db.open())
    {
        setCursor(Qt::ArrowCursor);
        QMessageBox::critical(this, tr("Error"),tr("Error opening database!"),
QMessageBox::Ok);
        return;
    }
    setCursor(Qt::ArrowCursor);
}

void MainWindow::closeDB(void)
```

```
{
    db.close();
}

void MainWindow::openTable(QString tableName)
{
    model = new QSqlRelationalTableModel(this);
    model->setJoinMode(QSqlRelationalTableModel::LeftJoin);
    model->setTable(tableName);
    if (tableName == "departments")
    {
        model->select();
        model->setHeaderData(0, Qt::Horizontal, tr("Id"));
        model->setHeaderData(1, Qt::Horizontal, tr("Name"));
    }
    else if (tableName == "authors")
    {
        model->setRelation(1, QSqlRelation("departments", "id", "name"));
        model->select();

        model->setHeaderData(0, Qt::Horizontal, tr("Id"));
        model->setHeaderData(1, Qt::Horizontal, tr("Department"));
        model->setHeaderData(2, Qt::Horizontal, tr("Name"));
        model->setHeaderData(3, Qt::Horizontal, tr("Phone"));
        model->setHeaderData(4, Qt::Horizontal, tr("E-mail"));
        ui->tableView->setItemDelegate(new QSqlRelationalDelegate(ui-
>tableView));
    }
    else if (tableName == "documents")
    {
```

```
        model->setRelation(2, QSqlRelation("doc_types", "id", "name"));
        model->setRelation(3, QSqlRelation("authors", "id", "name"));
        model->select();

        model->setHeaderData(0, Qt::Horizontal, tr("Id"));
        model->setHeaderData(1, Qt::Horizontal, tr("Name"));
        model->setHeaderData(2, Qt::Horizontal, tr("Type"));
        model->setHeaderData(3, Qt::Horizontal, tr("Author"));
        model->setHeaderData(4, Qt::Horizontal, tr("Document"));
        ui->tableView->setItemDelegate(new QSqlRelationalDelegate(ui->tableView));
    }
    else if (tableName == "doc_types")
    {
        model->select();

        model->setHeaderData(0, Qt::Horizontal, tr("Id"));
        model->setHeaderData(1, Qt::Horizontal, tr("Name"));
    }
    model->setEditStrategy(QSqlTableModel::OnRowChange);
//    model->setEditStrategy(QSqlTableModel::OnManualSubmit);
    ui->tableView->setModel(model);
    ui->tableView->setColumnHidden(0, true);
    ui->tableView->resizeColumnsToContents();
    //    ui->tableView->horizontalHeader()->resizeSection(1, 170);

    db.transaction();

    ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection);
    ui->tableView->setCurrentIndex(ui->tableView->model()->index(0, 0));
```

```
    //checkButtons();
    connect(ui->tableView->model(), SIGNAL(dataChanged(const
QModelIndex&, const QModelIndex&)), this,
SLOT(slotOnDataChanged(const QModelIndex&, const QModelIndex&)));
}

void MainWindow::slotOnDataChanged(const QModelIndex&, const
QModelIndex&)
{
//    isDataChanged = true;
//    checkButtons();
}

void MainWindow::slotAddRow(void)
{
    ui->tableView->model()->insertRow(ui->tableView->model()-
>rowCount());
//    isDataChanged = true;
//    checkButtons();
}

void MainWindow::slotDelRow(void)
{
    //int id = ui->tableView->model()->data(ui->tableView->model()-
>index(ui->tableView->selectionModel()->currentIndex().row(),0)).toInt();
    if (QMessageBox::question(this,tr("Query"), tr("Are you sure?"),tr("Yes"),
tr("No")))
        return;

    // Каскадное удаление
```

```
//    cascadeRemove(id);
    ui->tableView->model()->removeRow(ui->tableView->selectionModel()-
>currentIndex().row());
//    isDataChanged = true;
//    checkButtons();
}

void MainWindow::slotSaveChanges(void)
{
    ui->tableView->setCurrentIndex(ui->tableView->model()->index(ui-
>tableView->selectionModel()->currentIndex().row(), 0));



    model->submitAll();
    db.commit();
    model->select();
//    isDataChanged = false;
//    checkButtons();
}

void MainWindow::slotCancelChanges(void)
{
    model->revertAll();
    db.rollback();
    model->select();
//    isDataChanged = false;
//    checkButtons();
}
```

```
void MainWindow::slotOpenDocuments(void)
{
   openTable("documents");
}

void MainWindow::slotOpenDepartments(void)
{
   openTable("departments");
}

void MainWindow::slotOpenDocTypes(void)
{
   openTable("doc_types");
}

void MainWindow::slotOpenAuthors(void)
{
   openTable("authors");
}
```